Numerical Analysis (1/7)
Errors in numerical analysis

University of Luxembourg - 2024

Master in Mathematics

Hadrien Beriot

Outline

1.1 Introduction

Problem-solving process

1. Develop a **mathematical model** - usually expressed by equations of some type - of a physical phenomenon or system of interest *
2. Develop **algorithms** to solve the equations numerically *
3. Implement the algorithms in computer **software** *
4. Run the software on a computer to **simulate** the physical process numerically
5. **Post-process** results in some comprehensible form such as graphical visualization
6. **Interpret and validate** the computed results, repeating any or all the preceding steps, if necessary

We focus here on Steps **2 & 3**

* Primary tool = literature reviews, reading prior art

"

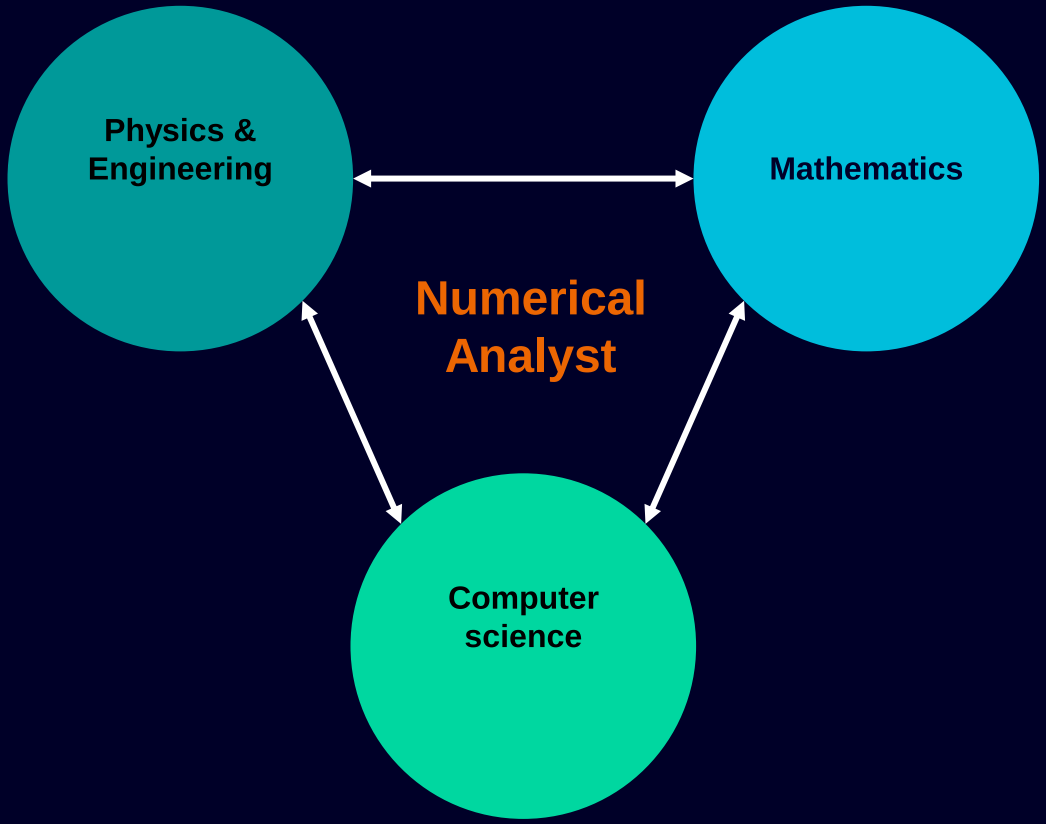
Strategy = replace the original problem with an easier one that has a close enough solution

Strategy

- Replacing infinite-dimensional spaces with finite-dimensional spaces
- Replacing infinite processes with finite processes, such as replacing integrals or
- infinite series with finite sums, or **derivatives with finite differences**
- Replacing differential equations with algebraic equations (finite difference)
- Replacing nonlinear problems with linear problems
- Replacing **high-order** systems with **low-order** systems
- Replacing complicated functions with simple functions, such as polynomials
- Replacing general matrices with matrices having a simpler form
- …
- Much of our effort → identifying transformations into simpler “solution-preserving” problems

Computer & scientists
Different flavors

- The **computer engineer** designs a new computer.
- The **computer scientist** develops the operating system and networking software for the computer.
- The **computational engineer or numerical analyst** uses computers and devises algorithms
  - to solve mathematical models
  - for complex systems
  - simulate behaviors
  - and analyze simulation output.



1.2 Sources of approximation

Sources of Approximation

- The **surface area of the Earth** might be computed using the formula

$$A = 4\pi r^2$$

- The use of this formula involves several approximations:
- Can you list some of them?

1. Earth is modeled as a sphere (idealization)
2. Radius, $r \approx 6370$ km, is based on empirical measurements and previous evaluations
3. Number π is given by an infinite limiting process, which must be truncated at some point
4. Input data, as well as results are rounded in a computer

The accuracy of the computed result depends on all these approximations.

Sources of Approximation

1. Modeling (friction/viscosity/air resistance)
2. Measurements (noise)
3. Previous computations (inaccurate inputs)
4. Truncation / discretization / computational: *from continuous to discrete*
5. **Rounding**: *computer floating-point representation*

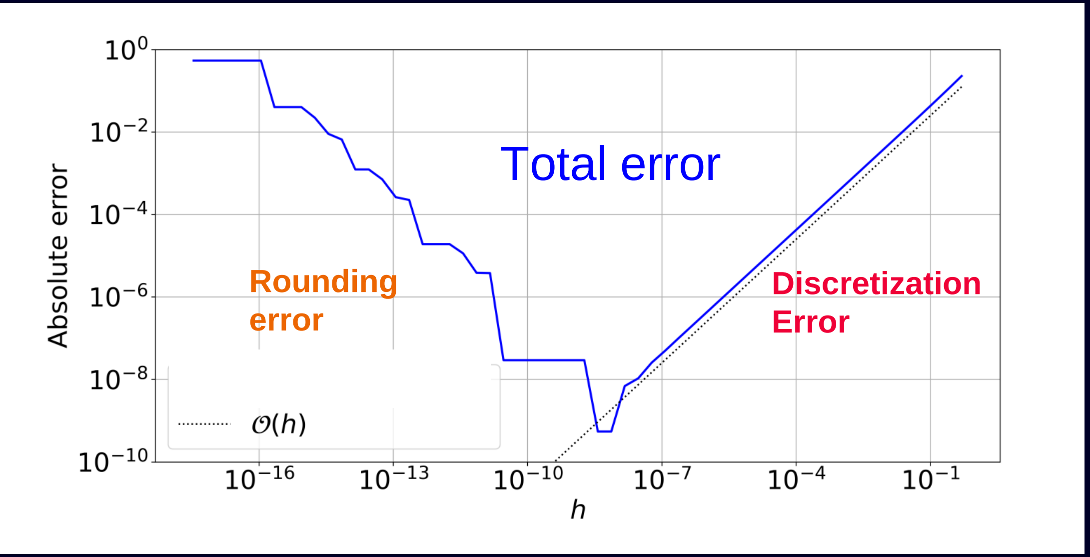
In this course, we focus on 4. and 5.

Round-off vs discretization errors
An example

- Derivative approximation (see exercise)

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} + O(h)$$

- The **discretization error** is linear with h, but when h is too small, we have **round-off errors***



*today's topic, computer floating-point representation

1.3 Representation of real numbers

Floating-point representation

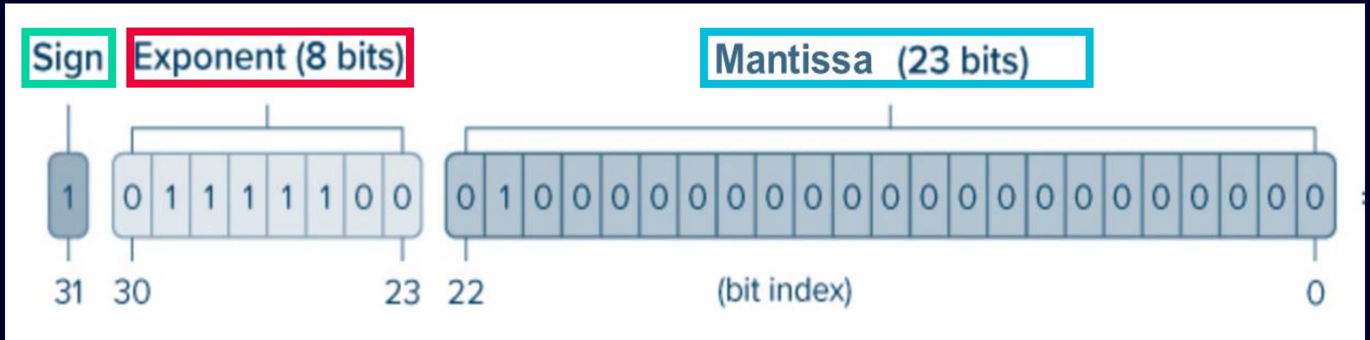
- A computer has a finite capacity and cannot store all real numbers (π ≈ 3.14159...)
- **Floating point** representation of $x \in R$

$$fl(x) = \pm m \times b^e, \qquad m = (\frac{d_0}{b^0} + \frac{d_1}{b^1} + \cdots + \frac{d_0}{b^{p-1}})$$

- m is the mantissa, e: exponent, b: basis, p: precision

- In base 10, **floating point ⬄ usual scientific notation**
- Let us choose $fl(x) = 152\,853.50, \ b = 10, \ p = 8, \ e = 5$
  - $152853.50 = 1.5285350 \times 10^5 = \left(1 \times 10^0 + 5 \times 10^{-1} + \cdots + 0 \times 10^{-7}\right) \times 10^5$

Floating-point representation
single precision (IEEE-754) – 32 bits



- **A floating-point system** is characterized by 4 values:
  - (b, p, U, L) base, precision, exponent upper and lower bound such as $L \leq e \leq U$
- Single precision format (32 bits storage) – IEEE 754 standard
  - $b = 2, p = 23, L = -126, U = 127$

$$(-1)^{sign} \times (1 + Mantissa) \times 2^{Exponent-127}$$

SIEMENS

Single/double precision
Example

- **Single precision** (np.float32):
  - 32 bits of which 23 for mantissa
  - $2^{\pm 127} \sim 3.4 \times 10^{\pm 38}$ range
  - $2^{-23} \sim$ 7 digits accuracy

- **Double precision** (np.float64):
  - 64 bits of which 52 for mantissa
  - $2^{\pm 1022} \sim 1.7 \times 10^{\pm 308}$ range
  - $2^{-52} \sim$ 16 digits accuracy

- **Example:**
  - You are solving $\boldsymbol{A}x = \boldsymbol{b}$ using an iterative solver (GMRES), but it's converging slowly.
  - You need to store 500 vectors, each representing a problem with 24 million degrees of freedom.
  - **Question:** How much RAM is required to store these vectors in double precision?

SIEMENS

1.4 Floating point arithmetic

Machine precision

- When rounding, two adjacent numbers are spaced by the **machine precision**

$$\eta = b^{-p/2}$$

  - For single precision (np.float32) we have $\eta = 2^{-23/2} \approx 1.19209 \times 10\text{^} - 7$
- We define the **round-off error** in absolute/relative terms as
  - Absolute: $|fl(x) - x| \leq \eta b^e$, (where e is the exponent and b =2)
  - Relative : $\left|\frac{fl(x)-x}{x}\right| \leq \eta$
- The usual calculus rules are altered with the floating-point representation

$$x \bullet y = (x \bullet y)(1 + \varepsilon), \qquad \bullet = (+,-,\times,\div), |\varepsilon| \leq \eta$$

- **Addition is not associative in this context** !

$$(a + b) + c \neq a + (b + c)$$

Absolute and relative spacing

We can check what is the relative and absolute spacing between two floating point numbers

```python
import numpy as np # import the numpy library

# distance between x and the nearest adjacent number
print(np.spacing(1e12)) # absolute spacing
print( (np.spacing(1e12)) / 1e12 ) # relative spacing
# print("{:e}".format(16**256)) # overflow

print(np.spacing(1e1)) # absolute spacing
print( (np.spacing(1e1)) / 1e1 ) # relative spacing
```

```
0.0001220703125
1.220703125e-16
1.7763568394002505e-15
1.7763568394002506e-16
```

Additioning floating point numbers

Be careful: the addition is not commutative in floating-point arithmetic !

```python
# Error in addition: (a+b) + c != a + (b+c) in floating point arithmetic
a = 2.3371258 * 10**(-5)
b = 3.3678429 * 10**1
c = -3.3677811 * 10**1

# amplification factors
ampli1 = (a+b) / (a+b+c)
ampli2 = (b+c) / (a+b+c)

print("{:.2f}".format(ampli1))
print("{:.2f}".format(ampli2))

t1 = round(a+b, 5) + c
t2 = a + round(b + c, 5)
print(f'(a+b) + c :{t1:.20f}')
print(f'a + (b+c) :{t2:.20f}')
```

```
52510.07
0.96
(a+b) + c :0.00063899999999250667
a + (b+c) :0.00064337125799999995
```


SIEMENS

1.5 Conditioning and stability

Stability

- Stability is an important concept in numerical analysis, and is found in different contexts:
  - Stability of numerical schemes (e.g. ODEs, finite differences),
  - Stability of numerical algorithms (e.g. linear systems),

- It is linked to the propagation of errors during the computation.

⚠ Stability has also meanings at the **continuous** level (PDE, dynamical systems).

Stability

- The forward error measures the difference between the computed and exact value $|\hat{y} - y|$
- The backward error is the error on the input: $|\hat{x} - x|$, for a perturbed output $\hat{y} = f(\hat{x})$
- If the backward error is “small”, we say the algorithm to be backward-stable

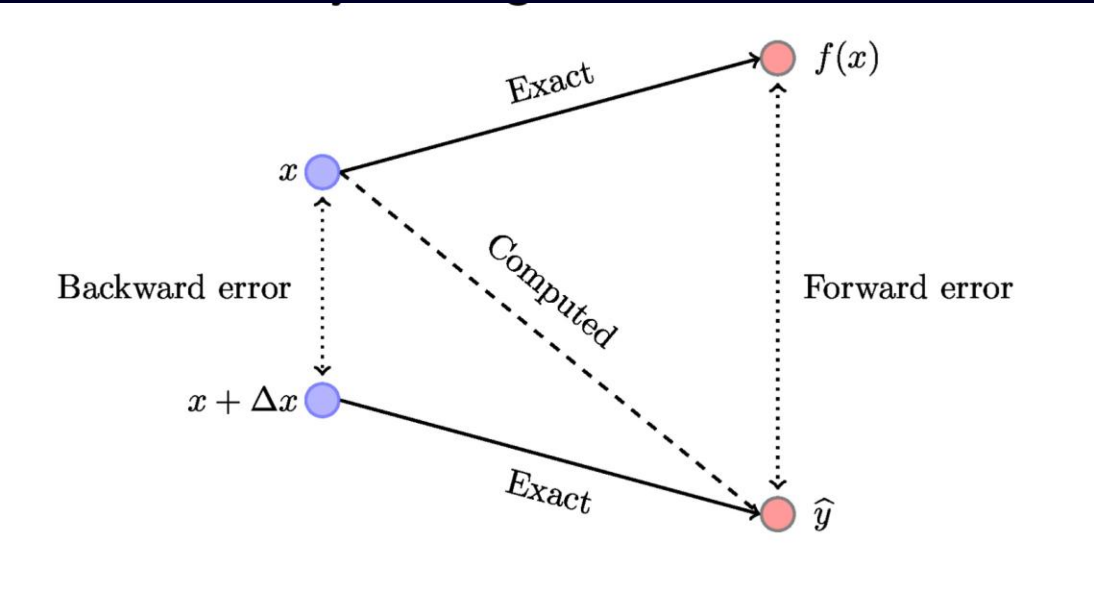


Illustration from Nicholas J. Higham blog

Conditioning

- The condition number is the ratio of a relative change in the output to a relative change in the
- Input. Condition number κ

$$\kappa = \left| \frac{f(\hat{x}) - f(x)/f(x)}{(\hat{x} - x)/x} \right|$$

- We have the relation

  |relative forward error| = Condition number × |relative backward error|
- It represents the sensitivity related to the input data.
- **Conditioning depends on the problem, stability depends on the algorithm**

- For a C^1 function differentiable function f: $\kappa = \left| \frac{\mathrm{f}'(\mathrm{x}) \Delta x / f(x)}{\Delta x / x} \right| = \left| \frac{\mathrm{x}\, \mathrm{f}'(\mathrm{x})}{\mathrm{f}(x)} \right|$

SIEMENS

Example $f(x) = 1 - x$

Conditioning

Good conditioning: small variations of data → small variations on the result.
For a function $f \in C^1$ the conditioning is

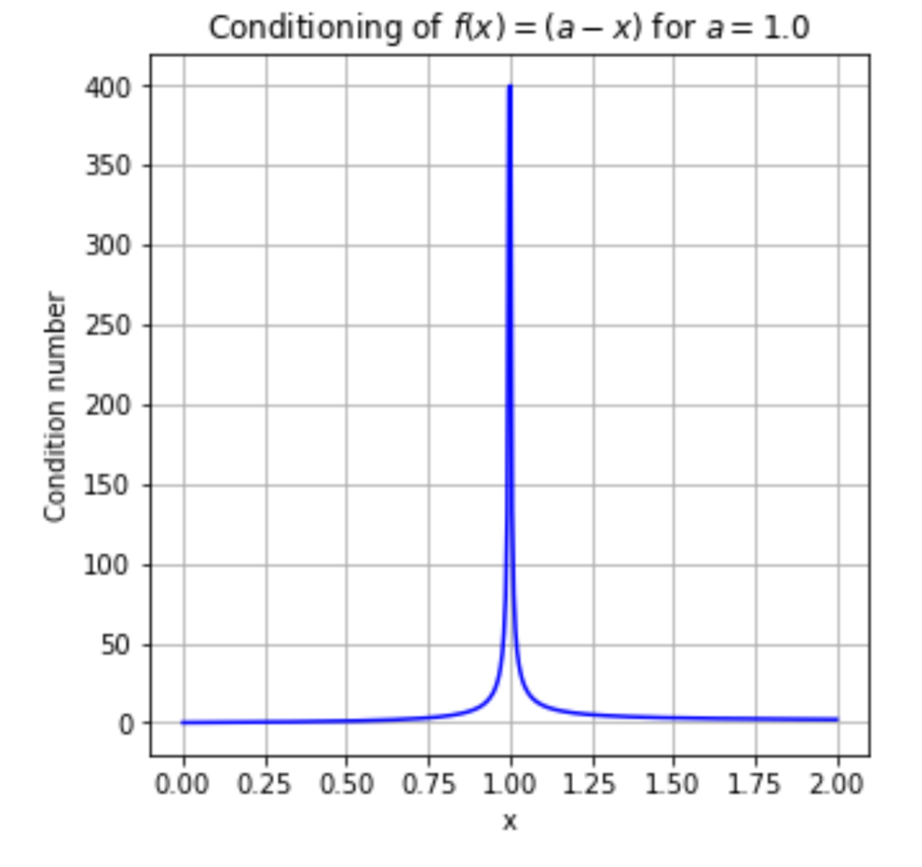
$$\mathrm{cond}(f)_x = \left|\frac{x f'(x)}{f(x)}\right|$$

For $f(x) = a - x$, the conditioning is large if x is close to a.

```
import matplotlib.pyplot as plt

a = 1.
x = np.linspace(a-1, a+1, 400)
# print(x)
cond_f = np.abs(-x/(a-x))
plt.figure(figsize=(5, 2.5))
plt.plot(x,cond_f, "b-")
plt.xlabel("x")
plt.ylabel("Condition number")
plt.grid()
plt.title("Conditioning of $f(x)=(a-x)$ for $a = %s$" %round(a,3))
```

```
Text(0.5, 1.0, 'Conditioning of $f(x)=(a-x)$ for $a = 1.0$')
```



SIEMENS

1.6 Summary

Summary

- All machines use a floating-point representation,
- We must be very careful during computations to avoid round-off errors,
- Conditioning depends on the problem, stability depends on the algorithm
- Designing numerically stable algorithms is in general not trivial

